EXHIBIT G

Lopez, Rula

From:	Brian Petry [bpetry@astutenetworks.com]
Sent:	Friday, August 30, 2002 1:24 PM
To:	patents@astutenetworks.com
Subject:	Info for provisional patent for "multi-protocol/multi-format applications all at the same time

Re: Provisional patent kick-off for:

"Multi-protocol/multi-format applications all at the same time"

From: Brian Petry

High-Level Design (HLD) References:
[1] Packet Processor (PP) 1.20: Intro and section 5 [2] Input Processing Unit (IPU): Just introductory sections [3] Dispatcher: Intro; sections about the event table and event types [4] Protocol Cluster; just the intro and memory map; 5 cores per cluster [5] Lookup Controller (LUC); flow key info, hash, hash table, timer control, split workspaces [6] Socket Memory Controller (SMC); page types, page pointer info, socket buffer configuration/initialization, socket buffer read and write commands

I thought this could be short, but after I started typing, I realized that maybe some further explanations and background might be useful. Please let me know if this is too verbose and I'll try to shorten the next one...

The general idea is that the chip architecture has special hardware and software processing units that allow for multiple protocols to be handled by the same chip at the same time. Example protocols: Fibre Channel, TCP/IP, API messages and proprietary/custom protocols. There are limitations in our current design as to how many protocols can live on the chip at once, but the invention shouldn't be limited to a certain number.

The work starts with our Packet Processor (PP) [1], which is the first relevant processing element to see a message coming from the outside. In the current design, we have 2 external interfaces that use SPI-4 (System Packet Interface). Multiple instances of PPs, 2 per SPI-4 handle the incoming messages. (the reason we have 2 is probably not relevant to multi-protocol/format; we need them to achieve a performance level). The messages (we try to call everything "events": packets, messages, frames, etc. once they're inside the chip) originate from other chips in a system and fall into multiple classes: network packets (different network formats, for example Fibre Channel vs. TCP/IP) and proprietary formats (formats that the chip defines and formats our customers can define). The importance of multiple classes I'll describe later. But within each class, we can have a number different event types that the PP can distinguish. In our design, any protocol/format can arrive on either SPI-4 interface. The main purpose of a PP is to identify the type of event, do some stateless processing on the event (like verify certain fields in a packet header have legal values) and extract an event-type-specific "flow key" (I'll describe the flow key below).

For instance, the PP can tell the difference between a TCP/IP packet and a UDP packet and an ICMP packet (all of these fall into a "TCP/IP network" class and the PP choses an appropriate "Event Type" that our chip uses (in the Dispatcher and processor cores). In this example, the "Flow Key" for a TCP/IP event type is: { IP Source Address(32 bits), IP Destination Address(32b), IP Protocol

Number(8b), TCP Source Port Number(16b), TCP Destination Port Number(16b), Source ID number(12b) }. The total number of bits in our flow key is 116; but that shouldn't be a limitation of the invention. For a different event type, the PP can format the flow key differently. Certain types of Fibre Channel frames, for instance, have a differently-formatted flow key.

The job of a flow key is to be a unique identifier of a state object that is required for later "stateful" processing of an event. As you probably know, a state-machine processing model basically takes in an event, examines the current state, takes some action based on the event type plus the current state, generates output events, updates the state, and then waits for a new event to arrive. You could say that our chip implements this model. The flow key is a virtual pointer to the state. The state in our chip is a small number of bytes of memory (in our case, 128Bytes to 2KB). With 116 bits it would not be practical to use the flow key as a real/physical memory pointer because it would require too much memory for the whole number-space of flow
keys: (2^116-1) * 2KB = 170141183460469231731687303715884103680 bytes of memory, so the chip implements a hash and hash-table lookup. This hash lookup algorithm by itself is probably not patentable because it has definitely been done before. But we have hardware engines to do the work. Anyway, the programmable context mechanisms will covered in a different Patent:
"Programmable context (flow state) area for the application."

In our current system, the flow key must be unique for the whole system, even different flow keys from different "classes" of events, because we implement only one hash table. But this should not be a limitation of the invention, which should claim a multiplicity of hash tables. The important thing is that the PP extracts the flow key, along with an event type, and sends it to the dispatcher.

Now I'll explain the reason for "classes" of messages (events). Our HLDs don't call them classes, but I picked that name here to help explain this.
Since other chips require to send fixed formatted events into our chip, the PP cannot distinguish the format sent from chip A from a format sent from chip B.
So, those are different classes. For instance, an Ethernet MAC chip would send us packets that are probably not distinguishable from a Fibre Channel MAC chip. But we still want to support hooking up to both chips at the same time.
Events are in the same class if the PP can distinguish them from each other.
Our chip handles different classes by requiring a configuration such that the other chips send each class on a different "logical" SPI-4 port number (the word "port" is bad, it should instead be called a "logical channel number"--it doesn't refer to a physical port). The SPI-4 port number is something specified by the SPI-4 standard (if you need the standard doc, let me know).
It's an identifier that allows different sources and destinations to be multiplexed over one SPI-4 interface. The PP simply vectors to different event-handling code for different SPI-4 port numbers, using a "static/pre-configured" distinction between the multiple formats.

In our chip, the PP is a programmable engine that is a custom processor built around a VLIW (Very Long Instruction Word) concept. So, different PP code can be loaded for different SPI-4 interfaces. Although our chip has some limitations about this, the invention should not be limited to the size of code store or the number of different code stores. I don't think we intend for the PP itself--the VLIW instruction set, capabilities, etc., to be on it's own patent. It you think it should/could be patented by itself, please let us know.

OK, enough about the PP, but one thing--the PP's are encapsulated inside Input Processing Units

(IPUs) [2]. Now that events are inside the chip, following the PP's "stateless" multi-protocol/multi-format processing, the dispatcher [3] takes over. The dispatcher's main purpose in the multi-protocol/multi-processing area is to identify what on-chip processor cores [4] are capable of processing the event and coordinate state-lookup (we call it "flow state" lookup), using the flow key, with the Lookup Controller [5]. The dispatcher is closely tied with the flow-director CAM (FDC) [6] to decide what flows are currently being processed on-chip (another patent will cover how the dispatcher and FDC make flow coherency work). The dispatcher uses the event type number to index a table that determines what processor cores are capable of processing what event types. This table is configurable and the cores' memories are loadable, lending to our ability to process various protocols and formats. The point is that different cores can be pre-configured to handle different protocols and event formats.

In our current chip, the Lookup Controller does not do much for multiple protocols/formats, since it has one global hash function, one hash lookup table and one configured flow state size. But future chips may be enhanced to allow for multiple hash functions, lookup tables and flow state sizes depending on the event type number, which would further enhance multi-protocols/formats capabilities. One protocol may benefit from a different hash function, a smaller (or larger) hash table and a different flow-state size, than some other protocol also being handled by the chip.

A complicated feature of the LUC and dispatcher is "split workspaces". It helps support multiple protocols by partitioning a flow state between one protocol and another by sending the flow state (we call it a "workspace" once it's inside the chip) to 2 processors. The idea is that each processor can work on a different protocol. But the protocols are related because they need to store state for the same logical "flow." An example of this is TCP and HTTP. TCP is one protocol and has a TCP flow state. HTTP is a different protocol, but it needs to store some information associated with the TCP flow state; it needs per-TCP-flow state to be saved. The relationship between the
2 protocols is usually 2 layers. In this example, TCP is a layer below HTTP in a "protocol stack." The dispatcher assigns two cores, a "protocol core"
and an "interface core" to work on a flow, and the LUC manages moving the 2 chunks of workspace between DDR and the cores. See Dispatcher section 1.4.3 and LUC 1.2.5.

Another thing the LUC does is process flow timers (this will be covered by another patent). How this relates to multi-protocol features is in the event type it uses for event expirations. When a timer expires in a workspace, it sends a flow-specific timeout expiration event to the dispatcher, which does its usual job of selecting an appropriate processor that's capable of handling the event number. Our current design has a flaw that the LUC timer expiration always sends the same event number so you might not find a description of this in the HLDs. In future versions of the chip, the event number will be selectable by the flow that has the timeout.

When a processor core gets an event and a state (if the event is stateful).
it has all the information it needs to implement the guts of a state machine. I didn't mention before that we also have other dispatcher-defined event type behaviors: stateless events, core-directed events and repeated-unicast (a cheap version of multicast) events. Note that the event number space can be assigned such that a single processor core can handle multiple protocols/formats.

One thing a core can do that is unique is to "chain" to a new flow. It can extract a flow key from the input event+state and issue a new event to the dispatcher, which in turn uses the FDC/LUC and the new event type to do another state lookup and chose another core. This is a feature of multi-protocol/format because one protocol-handler can decide to invoke another protocol-handler and the hardware queues and engines in our chip handle the work of looking-up the new flow state and assigning another processor core in an efficient manner.

The Socket Memory Controller (SMC) also helps with multiple packets/formats.
The SMC administers per-flow buffers: one send buffer and one receive buffer.
It organizes physical memory into pools based on page sizes. Currently, our design implements 4 page sizes, and a different protocol choses what page size fits best for it's buffering. For instance, a small-packet-sized protocol would benefit from small buffers and small page sizes, but a large-packet-sized protocol would benefit from large buffer. The chip can handle both types of protocols at once. Our current design is limited to 4 page sizes, but that shouldn't limit the invention. The SMC does all the work about organizing a list of pages so that software on a core can command the SMC to store and retrieve data based on a virtual buffer offset number. Work that was traditionally done in software for software-defined protocols is handled by the SMC. Specifics about how the SMC works to perform buffer sequencing and flow control (credit) management will be covered by a different patent.

-Brian